Journal of Computer and System Sciences **58**, 407–427 (1999)
Article ID jcss.1999.1626, available online at http://www.idealibrary.com on IDEAL®

On the Complexity of Database Queries

Christos H. Papadimitriou

Division of Computer Science, University of California, Berkeley, Berkeley, California 94720
E-mail: christos@cs.berkeley.edu

and

Mihalis Yannakakis

Bell Laboratories, Lucent Technologies, Murray Hill, New Jersey 07974
E-mail: mihalis@research.bell-labs.com

Received January 2, 1998; revised September 25, 1998

We revisit the issue of the complexity of database queries, in the light of the recent *parametric* refinement of complexity theory. We show that, if the query size (or the number of variables in the query) is considered as a parameter, then the relational calculus and its fragments (conjunctive queries, positive queries) are classified at appropriate levels of the so-called W hierarchy of Downey and Fellows. These results strongly suggest that the query size is inherently in the exponent of the data complexity of any query evaluation algorithm, with the implication becoming stronger as the expressibility of the query language increases. On the positive side, we show that this exponential dependence can be avoided for the extension of acyclic queries with $\neq$ (but not $<$) inequalities.

1. INTRODUCTION

The complexity of query languages has been—next to expressibility—one of the main preoccupations of database theory ever since the paper by Chandra and Merlin 20 years ago [5]; see [8, 2] for extensive overviews of the subject. It has been noted rather early [16] that, when considering the complexity of evaluating a query on an instance, one has to distinguish between two kinds of complexity: *Data complexity* is the complexity of evaluating a query on a database instance, *when the query is fixed*, and we express the complexity as a function of the size of the database. The other, called *combined complexity*, considers both the query and the database instance as input variables; the combined complexity of a query language is typically one exponential higher than data complexity.[1] Of the two, data complexity is widely regarded as more meaningful and relevant to database

[1] A third kind, *expression complexity* assumes that the database instance is fixed, and is rarely differentiated from the combined complexity.

0022-0000/99 $30.00

research, since the query is typically much smaller than the database, and hence, the query size can be productively assumed to be fixed by comparison.

For a broad range of important query languages (relational languages like conjunctive queries, first-order, i.e., full relational algebra and calculus, Datalog, fixpoint logic, as well as constraint languages, i.e., extensions with constraints such as arithmetic comparisons, linear and polynomial inequalities) data complexity predicts that the query evaluation problem is perfectly tractable; the complexity classes spanned by these query languages range from AC_0 to P, well within what is considered satisfactory in complexity theory. These tractability results are often quoted in the literature to suggest that the corresponding computational problems are tractable, well-understood, solved, under control. This implication is based on the thesis, broadly accepted in the theory of algorithms, that, as a rule, polynomial algorithms that arise in practice are usually fast, practical, with tolerable constant coefficient and reasonable exponents. Is this conclusion justified in the context of database query processing?

It seems to us that neither of the two notions of complexity is completely satisfactory. On the one hand, combined complexity is rather restrictive because it treats queries and databases as part of the input the same way, even though the size q of queries is typically orders of magnitude smaller than the size n of the database. Indeed it is for this reason that the study of the complexity of query languages has mostly concentrated on data complexity. However, on the other hand, polynomial time in the context of data complexity means time n^q, and in fact the known algorithms that place the above-mentioned languages in P have precisely such a running time. Moreover, in the case of fixpoint logic, this is known to be inherently unavoidable [16]. Even though $q \ll n$, it is not reasonable to consider q fixed, because even for small values of q, a running time of n^q hardly qualifies as tractable, especially in view of the fact that n is typically huge. What should the notion of complexity be then? What we would like to have is a running time in which n is not raised to a power that depends on q, i.e., the dependence on n is of the form n^c where c is a constant independent of the query (and hopefully very small).

Let us draw an analogy with the computer-aided verification area. The basic problem there is the model checking problem: does a given program P (the "model") satisfy a desired property ϕ (expressed in some specification language such as LTL, propositional linear temporal logic). There have been significant advances in recent years in the development of algorithms and tools in this area, especially for finite-state programs, which cover an important set of critical applications. The model-checking problem for finite state programs P and LTL specifications ϕ is PSPACE-complete. However, usually specifications are rather small (like queries) and programs are quite large (like databases). Fortunately, it turns out that the model-checking problem for LTL specification ϕ and program P can be solved in time exponential in $|\phi|$ and linear in $|P|$ [11].

Can we hope for such algorithms in the query evaluation of important query languages? *What are natural classes of queries that possess this type of algorithms*? These are the questions we seek to address.

Parametric complexity provides a framework to examine these problems. We now know that there is a class of reasonably natural problems that do not fall into

this mold: *parametric problems*, such as "does graph G have a clique of size k?" This problem, like many others, is currently solvable only by algorithms of complexity n^k. Query evaluation problems lie ominously within the scope of this category, with query length being the obvious analog of k in the parametric clique problem above. Researchers in complexity have recently developed a theory of *limited nondeterminism* and *fixed-parameter tractability* [4, 13, 6] which seeks to make important distinctions, along the lines suggested above, between problems below NP.

In particular, parametric problems with input, say, (G, k) which are solvable in polynomial time when k is fixed, can be subdivided into two broad categories: Those for which the polynomial is of the form $n^{f(k)}$—i.e., "has k in the exponent"—and those for which it is of the form $g(k)\, n^c$ for some constant c, called respectively *parametrically* (*or fixed-parameter*) *intractable* and *tractable*. It is of great interest to distinguish between these two categories and to develop rigorous tools that classify problems with respect to them. Downey and Fellows have introduced a sequence of complexity classes of parametric problems, collectively called *the W hierarchy*, which capture reasonably well this important issue [6]. The classes of the W hierarchy are indexed by the numbers 1, 2, ..., plus two limiting classes W[SAT] and W[P]. These classes are quite rich in complete problems; the higher the W class, the less likely that the problem has a polynomial algorithm with time bound of the form $g(k)\, n^c$.

A point of this paper is that parametric complexity theory is a productive framework for studying the complexity of query languages, which puts the well-known tractability results of the query languages mentioned above under a different perspective, one that is perhaps more realistic and less confusing and misleading. In particular, we prove that the parametric versions of the query evaluation problem for conjunctive queries, positive queries, and first-order queries (i.e., relational algebra and calculus) are hard for higher and higher levels of the W hierarchy. Therefore, it is likely that any algorithm for the corresponding query languages must have the parameter inherently in the exponent; furthermore, this likelihood increases measurably with the expressibility of the language. At present, this is only a "likelihood" and not a "proof," because proving that these languages are indeed not parametrically tractable would imply that $P \neq NP$ and $P \neq PSPACE$ resolving longstanding conjectures. For languages with recursion, like fixpoint logic and Datalog, there is, however, no such obstacle and parametric intractability is provable: Vardi showed already in [16] that there are fixpoint queries (and the proof can be adapted for Datalog) such that the query size must inherently appear in the exponent.

We analyse the complexity of relational queries for two types of parameters: the query size q and the number of variables v that appear in the query. The latter parameter is motivated by recent work of Vardi [17], who studied the complexity of queries assuming that the number of variables v is fixed, while the size of the query can grow along with the database. He found that this assumption brings the combined complexity closer to data complexity, namely polynomial time for the above languages, although the polynomial now has v instead of q in the exponent of n. Our analysis for the two parameters yields generally similar results (with some subtle differences).

Finally, we show a positive result which generalizes the main tractability result known so far in database theory, namely, that acyclic conjunctive queries can be evaluated efficiently (even with respect to combined complexity). We show that the extension of acyclic queries *with inequalities* (conjuncts of the form $x \neq y$) is parametrically tractable, in that the queries can be evaluated in time almost linear in the size of the database and the output, and exponential in the size of the query or the number of variables (this exponential dependence on the parameter is unavoidable, as the inequalities turn the combined complexity of the problem from polynomial to NP-complete). Trying to extend this further to $<$ constraints leads, however, to parametric hardness.

In the next section we give the necessary definitions from the (evolving) field of parametric complexity. In Section 3 we give the necessary definitions for applying this theory to query problems. In Section 4 we prove our classification results. Finally, in Section 5 we discuss acyclic queries with inequalities.

2. PARAMETRIC COMPLEXITY THEORY

We introduce next the main concepts from the complexity theory of parametric problems. Our definitions generally follow [6]. A *parametric problem* is a set L of pairs (x, k), where x is a string and k is an integer parameter. A parametric problem is called a *fixed parameter* (*f.p.*) *tractable* if there is an algorithm A that determines whether $(x, k) \in L$ in time bounded by a function of the form $f(k) \cdot |x|^c$ for some constant c; we will say that A runs in f.p. polynomial time.

Several NP-complete problems when supplied with a meaningful, natural parameter yield parametric problems that are f.p. tractable. Examples: Given a graph and k pairs of nodes, are there node-disjoint paths between all pairs of nodes? [14] Given a graph and an integer k, is there a path of length k in the graph? [12, 3] Both problems, and many others like them, have algorithms with running time $f(k) \cdot n^c$, where n is the input size and c is a constant.

In contrast, several other NP-complete problems do not seem to be tractable when considered as parametric problems with the natural parameter; examples include important problems such as clique, dominating set, bandwidth, etc. All these problems are solvable in time growing as $O(n^k)$ or a similar function, where n is the input length and k is the parameter (desired clique size, dominating set size, and bandwidth size in the three examples above), and, despite considerable effort to this end, no algorithm for each one of them is known with running time without k appearing in the exponent.

It would be very interesting to develop a refinement of NP-completeness theory that anticipates this sophisticated form of apparent intractability. Such a theory has been emerging from the work of many people, but most recently and notably Downey and Fellows [6]. There appears to be a *hierarchy* of parametric problems, called the *W hierarchy*, which classifies many of these problems. We first need to introduce an appropriate notion of reduction (in the literature one finds several more general kinds of reductions, but the one given next turns out to be the more useful one, certainly for the purposes of this paper).

A *parametric reduction* between two parametric problems A and B is an algorithm which solves any instance (x, k) of A using the answers to several instances (y_i, ℓ_i) of B, where (1) all ℓ_i are upper bounded by $g(k)$ (independent of x) for some function g, and (2) the instances of B and the final answer can be constructed in time $h(k)|x|^s$, for some function h and integer s. Such reductions are often *parametric transformations*, producing for any instance (x, k) of A an equivalent instance (y, ℓ) of B, and running in time $h(k)|x|^s$ for some function h and integer s.

Consider a Boolean circuit with AND, OR, and NOT gates and one output. We allow OR and AND gates of unbounded fan-in and fan-out. The circuit is *monotone* if it has no NOT gates. The *depth* of a circuit is the longest path from any input to the output (where as is the usual convention, we do not count any NOT gates applied to inputs). Let us now define `depth-t weighted satisfiability` for $t > 1$, to be the following parametric problem: Given a depth-t circuit C and an integer k, is there a setting of the inputs of C *with* k *inputs set to* 1 so that the output of C is 1? For $t = 1$ we require that the given circuit C be a 3-CNF formula, that is, a conjunction of clauses, where each clause is a disjunction of at most three literals (input variables or their negations). Also, the (unrestricted) `weighted circuit satisfiability` is the same problem with no depth restriction: Given a circuit C and an integer k, is there a setting of the inputs of C with k inputs set to 1, so that the output of C is 1? Finally, the `weighted formula satisfiability` problem is the case where the circuit has fan-out 1 (i.e., it is a Boolean formula).

We are now ready to define the classes in the W hierarchy; we give the definition in terms of their complete problems. We define $W[t]$ to be the set of all parametric problems that reduce to `depth-t weighted satisfiability`. The limiting classes W[SAT] and W[P], are the sets of all parametric problems that reduce respectively to `weighted formula` and `weighted circuit satisfiability`, with unlimited depth. In [6] it is pointed out that these classes have many natural complete problems, under parametric reductions. For example, `clique` is W[1]-complete and `dominating set` is W[2]-complete, while `bandwidth` is W[t]-hard for all $t > 0$. If a parametric problem is W[t]-hard, this means that it is very unlikely that it is tractable. The higher the t for which W[t]-hardness is proved (or, at the limit, W[P]-hardness), the stronger the implication of intractability.

It should be noted that the W hierarchy, as defined in [6], does not appear to have the classification power of, say, NP-completeness theory and of the polynomial hierarchy, in that many natural problems are only partially classified, that is, proved hard for one class and contained in another, higher one (or, as in the case of `bandwidth`, W[t]-hard for all $t > 0$ but not known to be in W[P]). This imperfect classification power is apparent in our results as well.

3. PARAMETRIC COMPLEXITY OF QUERY LANGUAGES

We review briefly first basic definitions on databases and queries. A *database* $d = \{D; R_1, ..., R_m\}$ consists of a domain D and a set of relations $R_1, ..., R_m$ over D. A *query* Q is a function that maps a database d to a relation $Q(d)$ (of certain arity)

over the same domain D. Queries are specified using *query languages*. A query language is capable of expressing a corresponding class of queries.

We will discuss in this paper the following languages (classes of queries): conjunctive queries, positive queries, first-order queries, and Datalog. Conjunctive queries correspond to relational calculus with conjunction and existential quantification (or equivalently relational algebra with selection, projection, join, and renaming). A *conjunctive query* is a query of the form $G = \{t_0 \mid \exists x_1 \cdots \exists x_k R_{i_1}(t_1) \wedge \cdots \wedge R_{i_s}(t_s)\}$, where G is the new relation defined by the query, t_0 is a tuple made of constants and variables $x_{k+1}, ..., x_m$, the R_{i_j} are database relations and the t_{i_j} are tuples of the appropriate arity consisting of constants and variables among $x_1, ..., x_m$. We will often write conjunctive queries using the standard rule notation $G(t_0) \leftarrow R_{i_1}(t_1), ..., R_{i_s}(t_s)$, which leaves implicit the existential quantification of the variables $x_1, ..., x_k$ that appear in the right-hand side (the body $R_{i_1}(t_1) \cdots R_{i_s}(t_s)$) but not in the left-hand side (the head $G(t_0)$). *Positive queries* add disjunction (union in algebra) to the list of operators; i.e., a positive query is of the form $G = \{t_0 \mid \phi\}$, where ϕ is a formula built from atoms $R_{i_j}(t_j)$ using $\exists$, $\wedge$, and $\vee$. *First-order queries* add negation (set difference in algebra), i.e., ϕ is an arbitrary first-order formula using the database relations. *Datalog* adds recursion to the positive queries; a Datalog query is a set of rules as above, using the relations of the database (called EDB relations), and new relations (called IDB relations), one of which is distinguished as the "goal" (output) relation of the query. We refer to the textbooks [15, 2] for a detailed exposition.

In the *evaluation problem* for a query Q, we are given database d and wish to compute $Q(d)$. In the *decision problem*, we are given in addition to the database d a tuple t and wish to decide if $t \in Q(d)$. When discussing the complexity of these problems, we assume a standard encoding of databases and queries. The complexity of query languages is usually measured in database theory via the decision problem. The *combined complexity* of a query language Λ is the complexity of the decision problem (set) $\{(Q, d, t) \mid Q \in \Lambda, t \in Q(d)\}$. The *data complexity* of a query language Λ is the complexity of the sets $\{(d, t) \mid t \in Q(d)\}$ for queries $Q \in \Lambda$; that is, the query is regarded as fixed. Thus, for example, the data complexity of a query language Λ is polynomial if there is a function $f: \Lambda \to N$ from queries to positive integers such that for every $Q \in \Lambda$ there is an algorithm which on input a database d of size n and a tuple t decides if $t \in Q(d)$ in time $O(n^{f(Q)})$.

In order to define the *parametric* complexity of query languages, we must first decide on the appropriate parameter to use. Two possible parameters come to mind: the *query size* q (the length of the string needed to express the query in Λ), and the *number of variables* v appearing in the query. Another relevant issue is whether we assume that the *schema* (set of relations and their arity) is fixed or can vary. The relationhip between the resulting four parametric problems (the query complexity problem above parameterized with v as parameter, or with q as parameter, each with fixed or variable schema) is as depicted in the partial order in Fig. 1.

PROPOSITION 1. *If one of the four parametric problems in Fig.* 1 *is hard for a class in the W hierarchy, then all problems above it are also hard. If a problem is in some class in the W hierarchy, then all problems below it are also in the same class.*

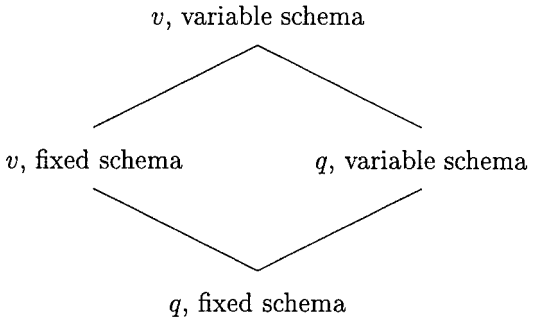

FIGURE 1

Proof. The identity map is a valid parametric reduction for all four arcs in the partial order. ▮

It turns out that the assumption on the schema makes no difference. We will show that the upper bounds hold for variable schema, while the lower bounds hold for a fixed schema.

4. A CLASSIFICATION OF QUERY LANGUAGES

We consider the following query languages: (1) conjunctive queries; (2) positive queries; (3) first-order queries. All these query languages are known to have data complexity AC_0 (which is contained in LOGSPACE and P).

THEOREM 1. *The parametric complexity of these query languages is classified as described in the following table, for both fixed and variable database schemas*:

	Parameter	
Query language	Query size q	Number of variables v
Conjunctive	W[1]-complete	W[1]-complete
Positive	W[1]-complete	W[SAT]-hard
First-order	W[t]-hard, all t	W[P]-hard

Proof. 1. *Conjunctive queries.* The lower bounds follow by a simple reduction from the `clique` problem, which is known to be W[1]-complete [6]. For any instance (G, k) of `clique` we construct a database consisting of one binary relation $G(\cdot, \cdot)$ (the graph). The query for parameter k is simply

$$P \leftarrow \bigwedge_{1 \leqslant i < j \leqslant k} G(x_i, x_j).$$

The goal proposition (0-ary relation) P is true iff G has a clique of size k. The query size is $q = O(k^2)$, while the number of variables is $v = k$, so this is a reduction to both problems. Note that this query just asks if the join of a set of binary relations is empty.

For the upper bounds, we give first the proof for the case of parameter q. We show how to transform the decision problem for conjunctive queries to the weighted satisfiability problem for Boolean formulas in 2-CNF. Consider a conjunctive query Q, a tuple t, and a database d; recall that the decision problem asks if $t \in Q(d)$. After substituting the constants of the tuple t in the query Q, the question is whether there is an instantiation of the variables of Q such that every atom of Q maps to a tuple in the database d. For every atom $a = R_i(\ )$ of the query Q and tuple s of the same relation R_i in the database d that is consistent with a we have a Boolean variable z_{as}, which intuitively stands for the statement "a maps to s."By "consistent" we mean that, if a has a constant in some column (attribute) then s has the same constant in that column, and if two entries of a are equal, then the corresponding entries of s are also equal. The set of clauses is defined as follows. For every atom a of Q and every pair of distinct tuples $s \neq s'$ we have a clause $\neg z_{as} \vee \neg z_{as'}$. For every pair of atoms a, a' and corresponding pair of tuples s, s' (including the case $s = s'$) such that a and a' have the same variable in columns j and j', respectively, but column j of s is not equal to column j' of s', we have a clause $\neg z_{as} \vee \neg z_{a's'}$. The parameter k of the 2CNF-satisfiability problem is the number of atoms of Q. It is not hard to see that there is an instantiation of the variables of Q that maps all the atoms to tuples of the database d if and only if the 2CNF Boolean formula has a satisfying assignment with k true variables. On the one hand, such an instantiation of Q's variables induces a truth assignment, where z_{as} is true if atom a of Q is mapped to tuple s of d; clearly the assignment has exactly k true variables and satisfies all the clauses. Conversely, a satisfying truth assignment with k true variables must have exactly one true variable z_{as} for each of the k atoms because of the first set of clauses, and it must induce an instantiation for the variables of Q that maps each atom a to a tuple s of d because of the second set of clauses.

Consider now the case where the parameter is v, the number of variables. As above, let Q be a conjunctive query (with the constants of a candidate answer tuple t substituted in the atoms), and d a database. In general, the size of Q, as well as the database schema (the number and arity of relations), may not be bounded by a function of v. We will transform the query and the database, so that the query is bounded by such a function and thereby reduce to the parameter q case. For a subset $S = \{x_{i1}, ..., x_{ir}\}$ of the variables, consider the set A_S of the atoms of Q that have exactly this set of variables. Our new query Q' has one atom $R_S(x_{i1}, ..., x_{ir})$ for each S such that A_S is nonempty, where R_S is a new relation scheme of arity r. Thus, Q' has size at most 2^v. The new database d' over these relations is constructed as follows. Each atom of Q in A_S is of the form $a = P(\tau)$, where P is a relation of the database d and τ is a tuple containing the variables $x_{i1}, ..., x_{ir}$, possibly with repetitions, and constants. From the database relation P we can compute an r-ary relation P_a that consists of those instantiations of the tuple of variables $(x_{i1}, ..., x_{ir})$ for which the atom a maps to a tuple of P. Relation P_a can be obtained from P by performing an appropriate selection according to the constants of the tuple τ and the equalities between different columns of τ, and then projecting to keep only one column for each variable. We then let R_S be the intersection of the relations P_a constructed in this manner for all atoms a of A_S. We do this for all

subsets S of variables that appear in atoms of Q. Clearly, the new database d' can be constructed in polynomial time from d and Q. By construction, an instantiation of the variables maps all the atoms of Q to tuples of the database d if and only if it maps all the tuples of Q' to tuples of d'.

2. *Positive queries.* In the parameter q case, the lower bound follows from part (1). The upper bound of W[1] follows from the well-known fact that we can transform a positive query into a union (disjunction) of (exponentially many in q) conjunctive queries; note that in this case we use the full power of parametric reductions, as opposed to transformations.[2]

In the parameter v case, the W[SAT] lower bound is by a reduction from the `weighted formula satisfiability` problem. Let ϕ be a Boolean formula in variables $x_1, ..., x_n$, and let k be the parameter. Our database d has binary relations $EQ = \{(i, i) \mid 1 \leqslant i \leqslant n)\}$ and $NEQ = \{(i, j) \mid 1 \leqslant i \neq j \leqslant n\}$. Let Q be the positive query $\exists y_1 \cdots \exists y_k[\bigwedge_{1 \leqslant i < j \leqslant k} NEQ(y_i, y_j)] \wedge \psi$, where ψ is obtained from ϕ by replacing every positive occurrence of a variable x_i by $\bigvee_{1 \leqslant j \leqslant k} EQ(i, y_j)$ and every negative occurrence of x_i by $\bigwedge_{1 \leqslant j \leqslant k} NEQ(i, y_j)$. Clearly, ϕ has a satisfying truth assignment with k true variables if and only if the query Q is true of the database d.

Note that in this construction, the query Q is in *prenex normal form*; i.e., all the quantifiers appear in the beginning. All queries can be put in prenex normal form, but this involves renaming of the variables, which in general increases their number and thus does not preserve the parameter v. It is worth mentioning that for queries in prenex normal form also the converse upper bound holds; that is, prenex positive queries under parameter v are in W[SAT] (and hence, W[SAT]-complete).

Let d be a database and Q a prenex positive query, $Q = \exists y_1 \cdots \exists y_k \psi$, where ψ is quantifier-free; we can assume again without loss of generality that Q is closed (has no free variables), by substituting the constants of a candidate answer tuple t into the query. We have a Boolean variable z_{ic} for $i = 1, ..., k$ and every constant c of the database domain. An instantiation of the variables $y_1, ..., y_k$ corresponds to the truth assignment that assigns $z_{ic} = 1$ iff y_i is mapped to the constant c. Construct a Boolean formula ϕ which is a conjunction of the clauses $\neg z_{ic} \vee \neg z_{ic'}$ for each $i = 1, ..., k$ and each pair $c \neq c'$ of distinct constants, conjuncted with the formula that is obtained from ψ by replacing every relational atom a by a corresponding subformula θ_a as follows. Let $a = R(\tau)$ where R is a database relation and τ a tuple of variables and constants. The formula θ_a is of the form $\bigvee_s \bigwedge_j z_{ic}$, where the disjunction ranges over all tuples s of R that agree with the tuple τ of the atom a in all the positions in which τ contains a constant, the conjunction ranges over all positions j in which τ does not have a constant but some variable y_i, and the corresponding conjunct is z_{ic}, where c is the constant of tuple s in position j. Thus,

[2] But we can also obtain a transformation, if so desired, as follows. Every conjunctive query Q_i in the disjunction can be transformed to a question of whether a given graph G_i has a clique of size k_i; for example, by doing the 2CNF construction of part (1) and having one node for each variable z_{as} and edges connecting all nodes that are not in a common clause. We can make all the parameters of the graphs G_i be equal to $k = \max\{k_i\}$ by adding $k - k_i$ new nodes to every G_i which are connected to each other and to all other nodes of G_i. Letting G be the disjoint union of these graphs, the decision problem for Q transforms to the question whether G has a clique of size k.

an instantiation to the variables $y_1, ..., y_k$ maps the atom a to a tuple of R iff the corresponding truth assignment satisfies θ_a. It follows that Q is true for the database d iff the Boolean formula ϕ has a satisfying truth assignment with k true variables.

3. *First-order queries.* The reduction is similar for the two cases. We present first the parameter v case. The reduction is from the `monotone weighted circuit satisfiability` problem, which is known to be W[P]-complete. That is, we are given a monotone circuit C (one that uses only AND and OR gates—without negation) and wish to determine whether it has a satisfying input with k true variables. We can assume that the given circuit alternates between OR and AND gates and that the output is an OR gate at level $2t$. Level 0 contains the input variables which we consider also as gates. The domain D consists of the gates of the circuit (i.e., there is one distinct constant for each gate), and the database d contains only a binary relation C, describing essentially the wiring diagram (dag) of the given circuit: The relation C contains the pairs (a, b) such that gate a has gate b as one of its inputs and contains, in addition, the pairs (c, c) for all 0-level gates c (i.e., input variables).

Define inductively the following sequence of first-order queries for the even (OR) levels of the circuit

$$\theta_0(x) = [\, C(x, x_1) \vee C(x, x_2) \vee \cdots \vee C(x, x_k)],$$
$$\theta_{2i}(x) = \exists y[\, C(x, y) \wedge \forall x(\neg C(y, x) \vee \theta_{2i-2}(x)].$$

Finally, the query is

$$Q = \exists x_1 \exists x_2 \cdots \exists x_k \theta_{2t}(o),$$

where o is the constant standing for the output gate. (Note: θ_{2t} is expanded fully using inductively the previous formulas in the sequence; the formula of the query has size $O(t+k)$ and uses $k+2$ variables.) Notice that a fixed schema (only a binary relation) is required.

Suppose that Q is true; thus, there is a mapping τ of the variables $x_1, ..., x_k$ to gates such that $\theta_{2t}(o)$ is true for this mapping. Let B be the set of input gates that are the image of one of the x_i's; clearly $|B| \leqslant k$ (the inequality may be strict because some variables x_i are mapped to noninput gates or to the same gate, but this does not matter). Consider the input τ' to the circuit which sets to 1 the inputs in B and to 0 the rest of the inputs. From the definition of $\theta_0(x)$, this formula is true for a 0-level gate x (under the given mapping τ of the existential variables) iff x in B, i.e., iff the corresponding circuit gate is 1. Inductively, it is easy to see that this holds for any OR gate; i.e., an OR gate x at level $2i$ is 1 under the input τ' if and only if $\theta_{2i}(x)$ is true for the mapping τ of the x_i's. This follows immediately from the definition of $\theta_{2i}(x)$, given that it holds for the level $2i-2$. Since $\theta_{2t}(o)$ is true for the mapping τ, the corresponding input τ' satisfies the circuit. If τ' has less than k true input variables, we can augment it arbitrarily to a satisfying input assignment with exactly k true inputs (because the circuit is monotone).

Conversely, suppose that the circuit has a satisfying input τ' with k variables set to 1. Map the existential variables $x_1, ..., x_k$ of the query to these input variables (0-level gates). Again, by the same inductive argument, an OR gate x at level $2i$ is 1 under the input τ if and only if $\theta_{2i}(x)$ is true for this mapping of the existential variables. Since the output gate o evaluates to 1, it follows that Q is true.

The parameter q case follows by the same reduction because the `monotone depth-t weighted circuit satisfiability` problem is W[t]-complete for every even t [6]. ▮

Downey, Fellows, and Taylor [7] independently studied recently similar questions (following our suggestion in [19] for investigating query complexity in this framework); they studied the parameter q case, proving similar results. They show W[1]-completeness for positive queries (they give the same simple reduction from clique, although their upper bound proof is much more complicated and involves a new characterization for W[1]). For first-order queries, they show completeness for another, more general class called AW[*], which extends W[t]. Briefly, the difference is that, instead of asking whether there exists a satisfying assignment with k true input variables for a given depth-t circuit C, the input variables of the circuit are partitioned into r sets (r is the parameter) $V_1, ..., V_r$, each with an associated quantifier $Q_i = \exists$ for odd i and $Q_i = \forall$ for even i, and the question is whether $\exists x_1 \in V_1, \forall x_2 \in V_2, ..., Q_r x_r \in V_r$ such that C accepts the input with $x_1, ..., x_r$ true and the other input variables false. It is easy also to adapt the reduction of Theorem 1 to this problem (using a more general quantifier prefix instead of just an existential one).

There is another class called AW[P] that extends W[P] in a similar manner, by allowing alternating quantification, instead of only existential (see [1] or the Appendix to [6]). It consists of the problems that can be reduced to the following parametric problem: Given a monotone circuit C (no depth restriction), whose input variables are partitioned into r sets $V_1, ..., V_r$, each with an associated quantifier $Q_i = \exists$ for odd i and $Q_i = \forall$ for even i, and integer k_i, the question is whether $\exists$ subset of size k_1 of V_1, $\forall$ subsets of size k_2 of $V_2, ..., Q_r$ subset of size k_r of V_r, such that C accepts the input with the input variables in these subsets assigned true and the other input variables false. The parameter k here is the sum of the k_i's. It is easy to adapt the reduction of Theorem 1 to show that the parameter v case of first-order queries is AW[P]-hard. The k query variables $x_1, ..., x_k$ are indexed as x_{ij}, $1 \leqslant i \leqslant r$, $1 \leqslant j \leqslant k_i$, corresponding to the sets $V_1, ..., V_r$. The quantifier prefix of the query becomes $Q_1 x_{11} \cdots Q_r x_{rk_r}$. The body is $[\theta_{2t}(o) \wedge {}_i \{\psi_i \mid Q_i = \exists\}] \vee \neg[\wedge_i \{\psi_i \mid Q_i = \forall\}]$, where ψ_i is a formula stating that the variables x_{ij} of the ith block are mapped to distinct input gates of V_i. For example, suppose that we pick an arbitrary representative c_i^* from each V_i and encode in the database the partition of the input variables of the circuit by a relation $P = \{(a, c_i^*) \mid a \in V_i,\ i = 1, ..., r\}$; then $\psi_i = \wedge_j [P(x_{ij}, c_i^*) \wedge (\wedge_{l \neq j} (\neg C(x_{ij}, x_{il}))]$. In the other direction, it is not clear that first-order queries under parameter v are in AW[P], for the same reason as we described in the case of positive queries: The definition of AW[P] (and the other W and AW classes) puts all the quantification in the front. While we can always transform a formula into prenex normal form (and in the parameter q case,

we can also put the quantifier-free part in disjunctive or conjunctive normal form at the cost exponential in q), in general this transformation does not preserve the number of v variables. (Of course, one could define yet another class with such "interspersed" quantification that reuses variables—but we will refrain from doing this in the absence of a wealth of natural complete problems.) For first-order queries in prenex normal form under parameter v we can show completeness for AW[SAT] (the alternating extension of W[SAT]), adapting along the same lines the proof of Theorem 1 for the prenex positive queries.

For recursive query languages like fixpoint logic and Datalog, the exponential dependence on the query size is actually provable without complexity assumptions. Vardi showed in [16] that there are fixpoint queries (and similarly, the same holds along the same lines for Datalog queries) of size polynomial in k that can be computed in time n^k, but not in n^{k-1}; i.e., the query size is *provably* inherently in the exponent in this case. This holds even if the database (EDB) relations have all fixed arity, although in the Datalog case the IDB relation does not (it has arity $O(k)$).

If we restrict all EDB and IDB relations in Datalog to have fixed arity (independent of the parameter), then Datalog is in W[1] (and thus W[1]-complete) for both parameters. To see this, use the ordinary bottom-up evaluation algorithm for Datalog that applies repeatedly the rules until a fixpoint is reached. If the maximum arity is r, then every IDB relation has at most n^r tuples and a fixpoint is reached in n^r stages. In each stage we need to compute for each rule a conjunctive query with at most v variables; by Theorem 1 the decision version of this problem is in W[1]. Thus, the evaluation of a Datalog query with fixed arity relations reduces to a polynomial number of W[1] problems, and hence, it is in W[1].

Can we prove for the first-order languages an unconditional result, as in the case of recursive languages? At present, this is not possible without resolving at the same time some of the classical conjectures in complexity theory. Recall that the combined complexity of conjunctive and positive queries is NP and of first-order queries is PSPACE. Hence, in the unlikely event that $P = NP$ or $P = PSPACE$, these query languages would be tractable. By contrast, the combined complexity of fixpoint logic and Datalog is EXPTIME-complete and it is known that $P \neq EXPTIME$ by the Time Hierarchy Theorem.

5. A TRACTABLE CASE

Is there a nontrivial class of queries that is parametrically tractable? Even some simple queries that involve joins are NP-complete in combined complexity and, as we saw, probably parametrically intractable as well. Acyclic joins with projection and selection form the major exception to this. We will show in this section a nontrivial extension of that result.

Consider a conjunctive query Q:

$$G(t_0) \leftarrow R_{i_1}(t_1), \ldots, R_{i_s}(t_s).$$

Let V be the set of variables of Q. Form a hypergraph H with set of nodes V and with a hyperedge for every atom in the body of Q which contains the variables that

occur in the atom. The query Q is called *acyclic* if the hypergraph H is acyclic (see, e.g., [15] for the definition and properties of acyclic hypergraphs). We can evaluate the query Q as follows. For every atom $R_{i_j}(t_j)$ in the body of Q, let U_j be the set of variables that occur in t_j. Compute a relation S_j over the set of attributes U_j whose tuples are the instantiations of the variables which map t_j to a tuple in R_{i_j}. S_j can be computed by performing appropriate selections and projection on R_{i_j}, namely $S_j = \pi_{U_j}\sigma_{F_j}(R_{i_j})$, where the selection F_j selects those tuples of R_{i_j} which (i) agree with t_j in positions where t_j has a constant and (ii) have equal entries in positions in which t_j has the same variable. The projection on U_j just keeps one copy for each variable. Let Z be the set of attributes corresponding to the variables of the tuple t_0 in the head. Compute $S^* = \pi_Z(S_1 \bowtie \cdots \bowtie S_s)$ from which we can easily construct the result of the query $Q(d) = \{\tau(t_0) \mid \tau \in S^*\}$. If Q is acyclic, this evaluation can be done in time polynomial in the size of the input database d and the output $Q(d)$ [18]. If we only want to check whether $Q(d)$ is empty or whether a specific given tuple t is in $Q(d)$, we can do it in time polynomial in the size of d (substitute the constants of t in the body of the rule and evaluate the resulting query which will be either empty or contain one tuple, t).

Suppose now that in the body of the conjunctive query we have, in addition to the relational atoms, inequality atoms $x_i \neq x_j$ or $x_i \neq c$ between the variables or variables and constants. In this case the query evaluation algorithm would normally include in the hypergraph also edges (x_i, x_j) corresponding to the inequalities between the variables (see [15]). However, inclusion of these edges destroys acyclicity even in very simple cases. Some examples: Find the employees that work on more than one project: $G(e) \leftarrow EP(e, p), EP(e, p'), p \neq p'$, where EP is the employee–project relation. Find the students that take courses outside their department: $G(s) \leftarrow SD(s, d), SC(s, c), CD(c, d'), d \neq d'$. Of course, in general we may have more complicated queries with multiple relations and the relations may not be binary (i.e., a genuine hypergraph).

Suppose that we have a conjunctive query with inequalities and that the hypergraph defined by considering only the relational atoms is acyclic. We call this an *acyclic query with inequalities*. Is the combined complexity still polynomial? Unfortunately, not: the problem becomes NP-complete. For example, the Hamiltonian path problem can be easily reduced to it. Given a graph (V, E), let Q be the query $G \leftarrow E(x_1, x_2), E(x_2, x_3), ..., E(x_{n-1}, x_n), x_1 \neq x_2, x_1 \neq x_3, ..., x_{n-1} \neq x_n$. The goal proposition (0-ary relation) G is true iff the graph is Hamiltonian. Here the query is as big as the database. However, in the more interesting case where the query is "small," the problem remains tractable, but now in the fixed parameter (f.p.) sense.

THEOREM 2. *The class of acyclic conjunctive queries with inequalities is f.p. tractable, both with respect to the query size and the number of variables as the parameter. Furthermore, we can evaluate such a query in f.p. polynomial time in the input and the output.*

A special case is the problem of finding simple paths of a specified length k in a graph. This problem was proved f.p. tractable by Monien [12], and an improved algorithm was given in [3] using an elegant "color-coding" (hashing) technique. Our algorithm combines this technique with acyclic query processing techniques.

The basic idea is to hash the domain D into a smaller domain (with the size bounded by the number of variables) and to use the hash values to check inequalities, while using the original values to check equality on the join attributes. Let Q be an acyclic query with inequalities, and let $H=(V, E)$ be its hypergraph. Partition the inequality atoms of Q into the set I_1 of atoms $x_i \neq x_j$ such that the variables x_i, x_j do not occur together in any hyperedge (relational atom), and the set I_2 of the remaining atoms ($x_i \neq c$ and $x_i \neq x_j$ such that x_i, x_j are in a common hyperedge). Let V_1 be the set of variables that occur in I_1 and let $k=|V_1|$. Let h be a function that maps the domain D to the set $\{1, ..., k\}$. Consider an instantiation τ of the variables. We say that τ is *consistent* with h if for every inequality $x_i \neq x_j$ of I_1 we have $h(\tau(x_i)) \neq h(\tau(x_j))$; clearly this implies also that $\tau(x_i) \neq \tau(x_j)$, but not necessarily vice versa. The instantiation τ is *satisfying* if it satisfies all the (relational and inequality) atoms in the body of Q. Let Θ_h be the set of all consistent satisfying instantiations, and let $Q_h(d)=\{\tau(t_0) \mid \tau \in \Theta_h\}$.

Fix a function $h: D \to \{1, ..., k\}$. We describe an f.p. polynomial time algorithm that decides whether there is a consistent satisfying instantiation τ and computes $Q_h(d)$. First, compute for each relational atom $R_{i_j}(t_j)$ of Q a corresponding relation $S_j=\pi_{U_j}\sigma_{F_j}(R_{i_j})$ over the set of attributes U_j (the variables that appear in t_j), where the selection F_j reflects as before (i) the constants that occur in the tuple t_j, (ii) the equalities between different positions of t_j, and in addition, it incorporates (iii) the inequality atoms $x_i \neq c$ such that $x_i \in U_j$ and (iv) the inequalities $x_i \neq x_l$ such that both x_i, x_l are in U_j.

Let V_1' be a set of new attributes corresponding to V_1. The domain of these new attributes is $\{1, ..., k\}$. If $X \subseteq V$ is a set of (original) variables, we use X' to denote the set of new attributes $\{x_i' \mid x_i \in X \cap V_1\}$. If t is a tuple over X, we can extend it to a tuple over XX' by letting $t[x_i']=h(t[x_i])$ for each $x_i' \in X'$. Extend in this manner each relation S_j to a relation S_j' over the set of attributes U_jU_j'; note that S_j' has the same number of tuples as S_j and the new attributes take values in $\{1, ..., k\}$. The algorithm is conceptually simple: For the emptiness problem, in essence what we will compute is the selection on inequalities $\{x_i' \neq x_l' \mid x_i \neq x_l \in I_1\}$ of the projection on V_1' of the join of the relations S_j'. This join is acyclic, so the projection can be computed by the algorithm of [18] in time polynomial in its size, which is at most k^k since the new attributes have domain of size k. The selections and projections can be pushed inside the join for further efficiency (though the worst case accounting will still have a factor k^k). We proceed with the details of the algorithm.

Let T be a join forest for H. Recall that this is a forest which has the hyperedges as its nodes, and with the property that for every attribute x_i, the set of nodes of T (i.e., hyperedges of H) that contain x_i form a connected subgraph (i.e., a subtree) T_i. We assume without loss of generality in the following that T is a tree (otherwise, for example, we can add additional edges to form a tree).

Root the tree at some node. For each node j of T, let $T[j]$ denote the subtree rooted at j, and let at($T[j]$) be the set of attributes that appear at the nodes of $T[j]$. From the definition of a join tree, if a variable appears in $T[j]$ but does not belong to U_j, then it must appear in exactly one proper subtree of $T[j]$ rooted at a child of j. Let W_j be the set of variables $x_i \in V_1 - U_j$ such that x_i appears in $T[j]$—hence in a unique proper subtree rooted at a child of node j—and there is

an inequality $x_i \neq x_l$ of I_1 such that x_l does not occur in the same proper subtree. In other words, node j separates the subtree T_i corresponding to the attribute x_i from the subtree T_l corresponding to the attribute x_l. Let $Y_j = U_j U'_j W'_j$.

LEMMA 1. *The attribute sets Y_j form an acyclic hypergraph with the same tree T as its join tree.*

Proof. The join tree property holds obviously for the original attributes $x_i \in V$ (because T is a join tree for the attribute sets U_j). For a new attribute $x'_i \in V'_1$, the set of nodes j such that Y_j contains x'_i consists of the subtree T_i (of nodes that contain x_i), along with the path from the root of T_i up to its lowest ancestor node w such that $T[w]$ (the subtree rooted at w) intersects all the subtrees T_l corresponding to attributes x_l that participate in inequality atoms $x_i \neq x_l$ with x_i. ∎

To test if $Q_h(d) = \varnothing$, we perform a bottom-up pass of the tree T as follows.

ALGORITHM 1 (Emptiness test).

1. Initialize for each node $j \in T$ a relation $P_j := S'_j$.

2. Process all the nodes except the root in bottom-up order of T as follows. To process node j of T with parent u, compute $P_u := \sigma_F(P_u \bowtie \pi_{Y_j \cap Y_u}(P_j))$, where F is the conjunction of the inequalities $x'_i \neq x'_l$ such that $x'_i \in Y_j - U'_u$ and x'_l belongs to the attribute set of P_u at this point but not to Y_j. If $P_u = \varnothing$ then quit and report $Q_h(d) = \varnothing$.

3. If all nodes are processed successfully, then report $Q_h(d) \neq \varnothing$.

LEMMA 2. *If Algorithm 1 quits in Step 2 then $Q_h(d) = \varnothing$. If it succeeds, then $Q_h(d) \neq \varnothing$, and the join of the P_u's is a relation over the attribute set VV'_1 that consists of all tuples $\tau\tau'_1$ such that τ is a satisfying instantiation that is consistent with h and τ'_1 is the extension of τ to V'_1.*

Proof. Consider the executions of Step 2. Let u be a node, let $C_u(t)$ be the set of its children that have been processed up to the tth execution of Step 2, and let $X_u(t) = U_u \cup \{\text{at}(T[m]) \mid m \in C_u(t)\}$ be the set of variables that occur in u and the subtrees $T[m]$ of the processed children $m \in C_u(t)$. As each child m of u gets processed, the relation P_u is augmented with some new attributes, namely the attributes in $Y_m - U'_u$; these are the attributes $x'_i \in W'_u$ such that the proper subtree of u that contains x_i is the subtree $T[m]$ rooted at the child m. If $P_u(t)$ denotes the relation P_u at his point, its set of attributes is $\text{at}(P_u(t)) = U_u U'_u \cup \{W'_u \cap Y_m \mid m \in C_u(t)\}$. Let $M_u(t)$ be the set of instantiations τ for the variables of $X_u(t)$ that satisfy all relational atoms and inequalities of I_2 corresponding to u and nodes of the subtrees $T[m]$, $m \in C_u(t)$ and that are consistent with h for the inequalities in I_1 involving only attributes of $X_u(t)$, and let $M'_u(t)$ be the extension of $M_u(t)$ to the set of attributes $X_u(t)\, X'_u(t)$.

Then we claim that the relation $P_u(t)$ at this point is equal to the projection of $M'_u(t)$ on $\text{at}(P_u(t))$. This can be shown by induction on the number t of executions

of Step 2. Initially, $C_u(0) = \varnothing$ for all u, $X_u(0) = U_u$, $M_u(0) = S_u$, and Step 1 initializes P_u appropriately to $S'_u = M'_u(0)$.

Assume that the claim holds up to the tth execution and that we process the next node j with parent u. The new set $X_u(t+1)$ is the union of the old set $X_u(t)$ with $X_j(t) = \text{at}(T[j])$. We claim that $M'_u(t+1) = \sigma_F(M'_u(t) \bowtie M'_j(t))$, where F is the conjunction of the inequalities $x'_i \neq x'_l$ such that $x'_i \in Y_j - U'_u$ (equivalently, $x_i \in X_j(t) - X_u(t)$) and $x'_l \in \text{at}(P_u(t)) - Y_j$ (equivalently, $x_l \in X_u(t) - X_j(t)$). Clearly, an instantiation τ of the variables of $X_u(t+1)$ satisfies all the relational atoms of u and the subtrees $T[m]$, $m \in C_u(t+1)$ of the processed children, iff it satisfies these atoms for $T[j]$ and the previously processed children $T[m]$, $m \in C_u(t)$. Also, if an inequality atom involves only variables from $X_u(t+1)$, then either (i) all its variables belong to $X_u(t)$, or (ii) to $X_j(t)$, or (iii) it is of the form $x_i \neq x_l$, where $x_i \in X_j(t) - X_u(t)$ and $x_l \in X_u(t) - X_j(t)$. In the first two cases, consistency of the instantiation with h for the inequality atom is checked within $M'_u(t)$ and $M'_j(t)$, respectively, and in the last case it is checked explicitly that $x'_i \neq x'_l$.

Note that the intersection of the attributes of $M'_u(t)$ and $M'_j(t)$ is contained in $U_u U'_u \cap U_j U'_j \subseteq Y_u \cap Y_j$. Since $M'_u(t+1) = \sigma_F(M'_u(t) \bowtie M'_j(t))$ it follows from the induction hypothesis that $P_u(t+1) = \sigma_F(P_u(t) \bowtie \pi_{Y_j \cap Y_u}(P_j(t)))$ is the projection of $M'_u(t+1)$, as claimed.

If the algorithm quits at some point t, then $P_u(t) = \varnothing$ which implies $M'_u(t) = \varnothing$, and thus there is no consistent satisfying instantiation. If all nodes are processed successfully, then each final P_u is a relation over the set of attributes Y_u, and their join is a relation over all the attributes VV'_1. Clearly, by the above claim, the join contains all the tuples $\tau\tau'_1$, where τ is a satisfying consistent instantiation and τ'_1 its extension to V'_1. The converse is also true; i.e., the join contains no more tuples, because obviously the variables of each relational atom or inequality of I_2 are contained in the corresponding Y_u, and for each inequality atom $x_i \neq x_l$ of I_1, the corresponding new variables x'_i, x'_l are contained in some common Y_u (namely, the Y set of the node u that is the least common ancestor of the subtrees T_i, T_l). ∎

To compute $Q_h(d)$ (if it is not empty), we proceed as follows. Algorithm 1 has taken care of all the inequalities. By Lemma 2, the join of the final P_u's gives us the set Θ_h of all the consistent satisfying instantiations of the variables, which yields the query answer $Q_h(d) = \{\tau(t_0) \mid \tau \in \Theta_h\}$. We do not actually want to compute the join because it could be exponentially larger than the answer and the input database. Let Z be the set of variables that appear in the tuple t_0 of the head. We only need to compute instead the projection $P^* = \pi_Z(P_1 \bowtie \cdots P_s)$; the query answer then is $Q_h(d) = \{\tau(t_0) \mid \tau \in P^*\}$ and has the same cardinality as P^*. By Lemma 1, the attribute sets of the P_j's form an acyclic hypergraph with T as a join tree. So the projection of the join can be computed in polynomial time. We do a two-pass algorithm over the tree: first a downward pass that removes dangling tuples via semijoins, followed by an upward pass that performs the joins and projections.

ALGORITHM 2 (Evaluation of $Q_h(d)$).

1. Process all the nodes except the root in top-down order of T. For each node j with parent u, set $P_j := P_j \ltimes P_u$.

2. Process all the nodes except the root in bottom-up order of T. For each node j with parent u, set $P_u := P_u \bowtie \pi_{Z_j}(P_j)$, where $Z_j = (Y_j \cap Y_u) \cup (Z \cap \text{at}(T[j]))$.

3. At the root r, compute $P^* := \pi_Z(P_r)$, and return $Q_h(d) := \{\tau(t_0) \mid \tau \in P^*\}$.

After Algorithm 1, the relation P_u of each node u is join-consistent with the relation P_j of each child j, i.e., $P_u = \pi_{Y_u}(P_u \bowtie P_j)$, because this is enforced when j gets processed, and after that point P_j stays the same and P_u can only lose tuples. Step 1 of Algorithm 2 enforces join consistency also in the opposite direction, i.e., $P_j = \pi_{Y_j}(P_u \bowtie P_j)$. After this, the relations are pairwise consistent and thus globally consistent, i.e., $P_u = \pi_{Y_u}(P_1 \cdots P_s)$ for all u (every tuple participates in the join). Step 2 extends for each node u the relation P_u with the attributes of Z that appear in the subtree $T[u]$, and Step 3 computes the answer $Q_h(d)$.

We have outlined so far how to determine for a given function h whether there is a satisfying instantiation that is consistent with h (i.e., whether $Q_h(d) \neq \varnothing$) and how to compute $Q_h(d)$. Given a query Q and database d, how do we pick h?

Suppose there is a satisfying instantiation τ for the given query Q and database d and let l be the number of distinct values assumed by the variables in V_1, i.e., $l = |\tau(V_1)|$. Then τ is consistent with at least a fraction $l!/l^k > e^{-k}$ of the functions h from D to $\{1, ..., k\}$. Thus, trying out a set of $O(e^k)$ random functions h will determine with high probability whether $Q(d) = \varnothing$. Formally, for every positive number c, if we run Algorithm 1 $c \cdot e^k$ times, each time choosing a random function h and at the end declaring $Q(d)$ to be empty iff all runs fail, then the answer is correct with probability at least $1 - e^{-c}$. Clearly, on the one hand, if a run for some h succeeds then $Q_h(d) \neq \varnothing$ and, hence, also $Q(d) \neq \varnothing$. On the other hand, if $Q(d)$ is not empty, there is a satisfying truth assignment τ; thus every run has probability at least e^{-k} that it will choose an h that is consistent with τ and succeed, so the probability that all runs fail is no more than $(1 - e^{-k})^{ce^k} \leqslant e^{-c}$.

For a deterministic algorithm and to compute $Q(d)$, we can use a k-perfect family F of hash functions, i.e., a family F which has the property that for every subset S of k (or less) elements of D, there is a $h \in F$ that hashes S into distinct values. One can construct such a family F with $2^{O(k)} \log |D|$ hash functions that can be evaluated in constant time (see [3] and the references therein). Then $Q(d) = \bigcup_{h \in F} Q_h(d)$.

We discuss now the time complexity of the algorithm. Fix a function h, and consider Algorithm 1. Each relation S_j (and its extension S'_j) has at most as many tuples as the corresponding database relation R_{i_j} and can be easily constructed from it. Thus, the initial versions of the relations P_j constructed in Step 1 of Algorithm 1 have in total size at most $q\,|d|$ and can be constructed in the same amount of time. In Step 2, each relation P_u is augmented with additional columns of W'_u as the children of u are processed; every tuple of the new P_u is derived from an old tuple and values for the new attributes, whose domain is $\{1, ..., k\}$. Thus, the number of tuples of P_u increases at most by a factor of $k^{|W_u|} \leqslant k^k$. Thus, the total size of all the P_j is bounded always by $qk^k\,|d|$. The joins of Step 2 can be performed, for example, by sorting the two relations on the join attributes and merging. Thus, the time for Step 2 is at most $\Sigma_u\,|P_u| \log |P_u|$ degree(u), and the complexity of

Algorithm 1 for a given h is bounded by $O(k^k qn \log n)$, where n is the size of the input database. Running the algorithm for several functions h, we can determine whether $Q(d)$ is empty (or whether a specific given tuple t belongs to $Q(d)$), in randomized time $O(g(v)\, qn \log n)$ with high probability, or deterministically in time $O(g(v)\, qn \log^2 n)$, where $g(v) = 2^{O(v \log v)}$.

Algorithm 2 performs the projection of an acyclic join of the relations P_u, which in total have size bounded by $qk^k n$. If $m = |Q(d)|$ is the size of the output, then each relation P_u grows at worst by a factor of m during the joins of Step 2 (see, e.g., [15, 19] for the details), and hence, Algorithm 2 for a given h takes time bounded by $O(k^k qmn \log n)$. Running the algorithm for the functions h of a k-perfect family we can compute $Q(d)$ in time $O(g(v)\, qmn \log^2 n)$, where $g(v) = 2^{O(v \log v)}$. This concludes the proof of Theorem 2. ∎

If the parameter is q, the query size, the same theorem holds in the case where, instead of a conjunction of inequalities in the body of the query, we have an arbitrary Boolean formula ϕ built from inequality atoms using $\vee$ and $\wedge$. The method is similar—we only sketch it briefly. We use again hash functions h and introduce new attributes for all the variables that appear in ϕ which we use to check the condition ϕ. The size k of the range of h is, in general, taken now to be the sum of the number of variables and the number of constants that appear in the inequalities of ϕ; clearly $k \leqslant q$. The main difference now is that we may not be able to push the selection on the inequality constraints down in the tree, as we did in the case of a conjunctive ϕ.

The same theorem is true if the parameter is v, the number of variables, provided that the atoms $x_i \neq c$ appear only conjunctively, i.e., if we have a formula ϕ that is a conjunction of inequalities $x_i \neq c$, together with an arbitrary formula built from inequality atoms $x_i \neq x_j$ using $\vee$ and $\wedge$. In this case, we can incorporate the inequalities $x_i \neq c$ directly in the computation of the relations S_j from the database relations R_{i_j} and proceed with the rest of the formula ϕ (which has no constants) as above. The range k of the hash function is still bounded by the parameter v. However, if the inequalities between variables and constants are combined arbitrarily using $\vee$ and $\wedge$, then in general the sum of the numbers of variables and constants that appear in ϕ may not be bounded by any function of v. In fact, in this case the problem is not anymore f.p. tractable with respect to the parameter v; it becomes W[SAT]-complete. The proof is as in Theorem 1 for the parameter v case of positive queries in prenex normal form (replacing in the hardness proof every equality $y = i$ by a conjunction of inequalities $\bigwedge_{c \in D - \{i\}} (y \neq c)$).

Comparison Constraints

Can we extend the result to acyclic conjunctive queries with comparisons ($<$ or $\leqslant$) between variables or variables and constants? Example: Find the employees that have higher salary than their manager $G(e) \leftarrow EM(e, m)$, $ES(e, s)$, $ES(m, s')$, $s' < s$. First, note that trivially any equality $x = y$ can be expressed as the conjunction of the two weak inequalities $x \leqslant y$ and $y \leqslant x$, so the question makes sense only if we first identify equal variables; otherwise, we can express trivially any

conjunctive query by a set of atoms with disjoint variables and equalities. Given a conjunctive query Q with a set C of comparison atoms, we must first determine if C is consistent and find the implied equalities between variables and constants, which we then collapse. This can be done (for dense orders) by forming a graph whose nodes are the variables and constants in C, with a directed arc $u \to w$ between two nodes u, w labeled $<$ or $\leqslant$ if C contains the corresponding constraint $u < w$ or $u \leqslant w$ or u, w are constants with $u < w$. The system is consistent iff there is no strongly connected component that contains a $<$ arc, and the implied equalities are that all nodes of the same strong component are equal (see, e.g., [10]). Let Q' be the resulting query after collapsing equal variables and constants of Q, and C' its set of comparison constraints (which is now acyclic). We say that the query Q with comparisons is acyclic if the hypergraph corresponding to the relational atoms in the body of Q' is acyclic. Can we evaluate such a query in f.p. polynomial time? Unfortunately, not.

THEOREM 3. *The class of acyclic conjunctive queries with comparisons is* $W[1]$*-complete with respect to both parameters q and v.*

Proof. Membership in W[1] follows from Theorem 1. For the hardness we reduce from the `clique` problem. Let (G, k) be an instance of the `clique` problem, where G has n nodes numbered $0, ..., n-1$, and assume for notational convenience that every node has a self-loop. For all edges (i, j) of G and for $b = 0, 1$, let $[i, j, b]$ denote the integer $(i+j)\, n^3 + |i-j|\, n^2 + bn + i$. We construct a database with two binary relations P, R. The relation P consists of the tuples $([i, j, 0], [i, j, 1])$ for all edges (i, j) of G. The relation R consists of the tuples $([i, j, 1], [i, j', 0])$ for all i, j, j'. The query Q is

$$S \leftarrow \bigwedge_{1 \leqslant i,\, j \leqslant k} P(x_{ij}, x'_{ij}),\ \bigwedge_{1 \leqslant i,\, j,\, l = j+1 \leqslant k} R(x'_{ij}, x_{il}),\ \bigwedge_{1 \leqslant i < j \leqslant k} x_{ij} < x_{ji} < x'_{ij}.$$

Note that the query uses only strict comparisons, and the graph of the comparisons is clearly acyclic. The hypergraph of the query is a graph that consists of paths with alternating P and R edges. There are k paths, where the ith path is $x_{i1}, x'_{i1}, x_{i2}, x'_{i2}, ..., x_{ik}, x'_{ik}$.

We claim that the goal proposition S is true (Q is nonempty) iff G has a clique of size k. Suppose first that G has a clique of size k consisting of the nodes $v_1, ..., v_k$ in that order. Then let $x_{ij} = [v_i, v_j, 0]$ and $x'_{ij} = [v_i, v_j, 1]$. From the definitions of P and R, the relational atoms of the query are satisfied. As for the comparisons, a straightforward substitution gives $x_{ji} - x_{ij} = v_j - v_i$; thus, $x_{ij} < x_{ji}$ for $i < j$ follows from $v_i < v_j$. Also, $x'_{ij} - x_{ji} = n + v_i - v_j > 0$.

Conversely, suppose that the query has a satisfying instantiation τ of the variables. Because of the P and R edges of the query graph, τ must map nodes on each path to triples with the same first component. Let $v_1, ..., v_k$ be the first components of the images of the nodes on the k paths (in that order); i.e., the nodes of the ith path are mapped to triples of the form $[v_i, \cdot, \cdot]$. From the definition of P and because of the atom $P(x_{ij}, x'_{ij})$, a variable x_{ij} must map to a tuple with first component v_i and third component 0, and the variable x'_{ij} must map to the tuple

with a first component also v_i, the same second component, and the third component 1. Consider two distinct indices $i<j$, and let the images of $x_{ij}, x'_{ij}, x_{ji}, x'_{ji}$ respectively be $[v_i, v'_j, 0]$, $[v_i, v'_j, 1]$, $[v_j, v'_i, 0]$, $[v_j, v'_i, 1]$. We will show that $v_i = v'_i$ and $v_j = v'_j$. The inequality $x_{ij} < x_{ji} < x'_{ij}$ implies $(v_i + v'_j)\, n^3 + |v_i - v'_j|\, n^2 + v_i < (v_j + v'_i) n^3 + |v_j - v'_i|\, n^2 + v_j < (v_i + v'_j)\, n^3 + |v_i - v'_j|\, n^2 + n + v_i$. From the leading n^3 terms in the three expressions we conclude that the coefficients are equal,

$$v_i + v'_j = v_j + v'_i. \tag{1}$$

Given that the n^3 terms equal, the next terms involving n^2 yield that their coefficients must be also equal,

$$|v_i - v'_j| = |v_j - v'_i|. \tag{2}$$

From the first inequality we get then also

$$v_i < v_j. \tag{3}$$

Equation (2) implies that either $v_i - v'_j = v_j - v'_i$ or $v_i - v'_j = v'_i - v_j$. The first case combined with (1) yields $v_i = v_j$, contradicting (3). Thus, we must have $v_i - v'_j = v'_i - v_j$, which combined with (1) implies that $v_i = v'_i$ and $v_j = v'_j$. That is, x_{ij} is mapped to $[v_i, v_j, 0]$ and x'_{ij} to $[v_i, v_j, 1]$. From the atom $P(x_{ij}, x'_{ij})$ and the definition of P, we conclude that there is an edge (v_i, v_j). Therefore, the nodes $v_1, ..., v_k$ are distinct and pairwise adjacent; thus they form a clique of size k. ∎

Note that the theorem holds even in rather restricted cases (for binary relations, path queries, only $<$ constraints, etc.).

ACKNOWLEDGMENTS

We appreciate the feedback from the PODS program committee and Moshe Vardi.

REFERENCES

1. K. Abrahamson, R. G. Downey, and M. R. Fellows, Fixed-parameter tractability and completeness IV: On Completeness for W[P] and PSPACE analogues, *Ann. Pure Appl. Logic* (1995), 235–276.
2. S. Abiteboul, R. Hull, and V. Vianu, "Foundations of Databases," Addison–Wesley, Reading, MA, 1995.
3. N. Alon, R. Yuster, and U. Zwick, Color-coding, *J. Assoc. Comput. Mach.* (1995), 844–856.
4. J. F. Buss and J. Goldsmith, Nondeterminism within P, *SIAM J. Comput.* (1993), 560–572.
5. A. K. Chandra and P. M. Merlin, Optimal implementation of conjunctive queries in relational databases, *in* "Proc. 9th ACM Symp. Theory of Comp., 1977," pp. 77–90.
6. R. G. Downey and M. R. Fellows, Fixed-parameter tractability and completeness I: Basic results, *SIAM J. Comp.* (1995), 873–921.
7. R. G. Downey, M. R. Fellows, and U. Taylor, "The Parameterized Complexity of Relational Database Queries and an Improved Characterization of W[1], Dec. 1996," Proc. Conf. Discrete Mathematics and Theoretical Computer Science, pp. 194–213.
8. P. C. Kanellakis, Elements of relational database theory, *in* "Handbook of Theoretical Computer Science" (J. Van Leeuwen, Ed.), pp. 1074–1156, Elsevier, Amsterdam, 1991.

9. P. C. Kanellakis, Constraint programming and database languages: A tutorial, *in* "Proc. 14th ACM Symp. Principles of Databases Sys., 1995," pp. 46–53.
10. A. Klug, On conjunctive queries containing inequalities, *J. Assoc. Comput. Mach.* (1988), 146–160.
11. O. Lichtenstein and A. Pnueli, "Checking that Finite State Concurrent Programs Satisfy their Specifications, 1985," Proc. 12th Annual ACM Symp. on Principles of Prog. Lang., pp. 97–107.
12. B. Monien, How to find long paths efficiently, *Ann. Disc. Math.* (1985), 239–254.
13. C. H. Papadimitriou and M. Yannakakis, On limited nondeterminism and the complexity of the VC dimension, *J. Comput. System Sci.* **53** (1996), 161–170.
14. N. Robertson and P. D. Seymour, Graph minors XIII: The disjoint paths problem.
15. J. D. Ullman, "Principles of Database and Knowledge Base Systems," Comput. Sci. Press, New York, 1988.
16. M. Y. Vardi, "The Complexity of Relational Query Languages, 1982," Proc. ACM Symp. Theory of Computing, pp. 137–146.
17. M. Y. Vardi, "On the Complexity of Bounded-Variable Queries, 1995," Proc. 14th ACM Symp. Principles of Database Sys., pp. 266–276.
18. M. Yannakakis, "Algorithms for Acyclic Database Schemes, 1981," Proc. 7th Intl. Conf. Very Large Data Bases, pp. 82–94.
19. M. Yannakakis, "Perspectives on Database Theory, 1995," Proc. 36th IEEE Symp. Foundations of Comp. Sc., pp. 224–246.